

---

# ActiveMQ

## 一、ActiveMQ 简介

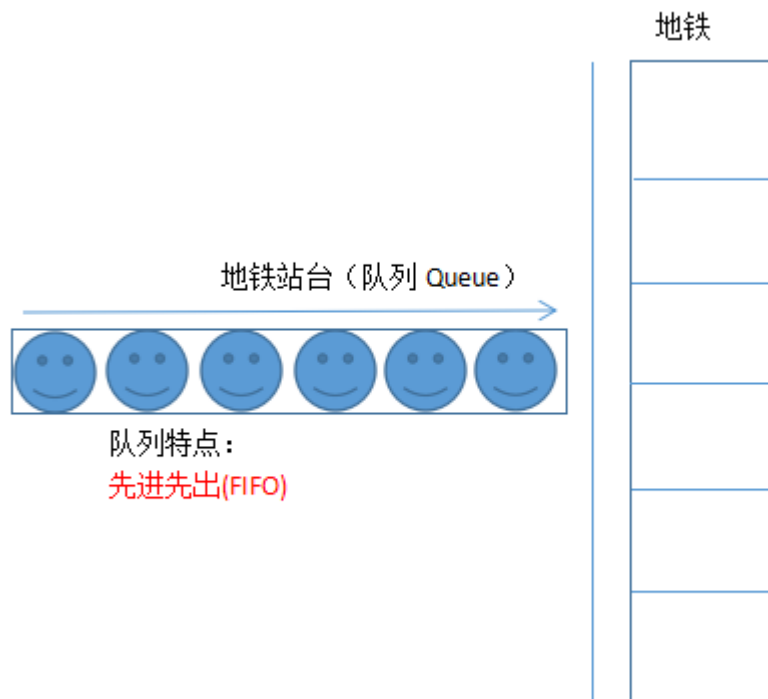
### 1 什么是 ActiveMQ

ActiveMQ 是 Apache 出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持 JMS1.1 和 J2EE 1.4 规范的 JMS Provider 实现，尽管 JMS 规范出台已经是很久的事情了，但是 JMS 在当今的 J2EE 应用中间仍然扮演着特殊的地位。

### 2 什么是消息

“消息”是在两台计算机间传送的数据单位。消息可以非常简单，例如只包含文本字符串；也可以更复杂，可能包含嵌入对象。

### 3 什么是队列



### 4 什么是消息队列

“消息队列”是在消息的传输过程中保存消息的容器。

---

## 5 常用消息服务应用

### 5.1 ActiveMQ

ActiveMQ 是 Apache 出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持 JMS1.1 和 J2EE 1.4 规范的 JMS Provider 实现。

### 5.2 RabbitMQ

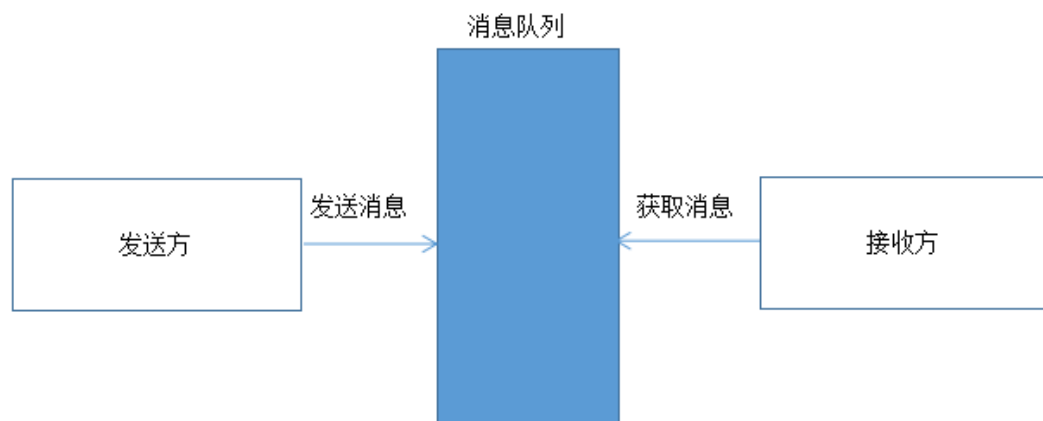
RabbitMQ 是一个在 AMQP 基础上完成的，可复用的企业消息系统。他遵循 Mozilla Public License 开源协议。开发语言为 Erlang。

### 5.3 RocketMQ

由阿里巴巴定义开发的一套消息队列应用服务。

## 二、 消息服务的应用场景

消息队列的主要特点是**异步处理**，主要目的是**减少请求响应时间和解耦**。所以主要的使用场景就是将比较耗时而且**不需要即时（同步）**返回结果的操作作为消息放入消息队列。同时由于使用了消息队列，只要保证消息格式不变，消息的发送方和接收方并不需要彼此联系，也不需要受对方的影响，即解耦和。



### 5.1 异步处理

#### 5.1.1 用户注册

用户注册流程：

- 1) 注册处理以及写数据库
- 2) 发送注册成功的手机短信
- 3) 发送注册成功的邮件信息

---

如果用消息中间件：则可以创建两个线程来做这些事情，直接发送消息给消息中间件，然后让邮件服务和短信服务自己去消息中间件里面去取消息，然后取到消息后再自己做对应的业务操作。就是这么方便

## 5.2 应用的解耦

### 5.2.1 订单处理

生成订单流程：

- 1) 在购物车中点击结算
- 2) 完成支付
- 3) 创建订单
- 4) 调用库存系统

订单完成后，订单系统并不去直接调用库存系统，而是发送消息到消息中间件，写入一个订单信息。库存系统自己去消息中间件上去获取，然后做发货处理，并更新库存，这样能够实现互联网型应用追求的快这一个属性。而库存系统读取订单后库存应用这个操作也是非常快的，所以有消息中间件对解耦来说也是一个不错的方向。

## 5.3 流量的削峰

### 5.3.1 秒杀功能

秒杀流程：

- 1) 用户点击秒杀
- 2) 发送请求到秒杀应用
- 3) 在请求秒杀应用之前将请求放入到消息队列
- 4) 秒杀应用从消息队列中获取请求并处理。

比如，系统举行秒杀活动，热门商品。流量蜂拥而至 100 件商品，10 万人挤进来怎么办？10 万秒杀的操作，放入消息队列。秒杀应用处理消息队列中的 10 万个请求中的前 100 个，其他的打回，通知失败。流量峰值控制在消息队列处，秒杀应用不会瞬间被怼死。

## 三、 JMS

### 1 什么是 JMS

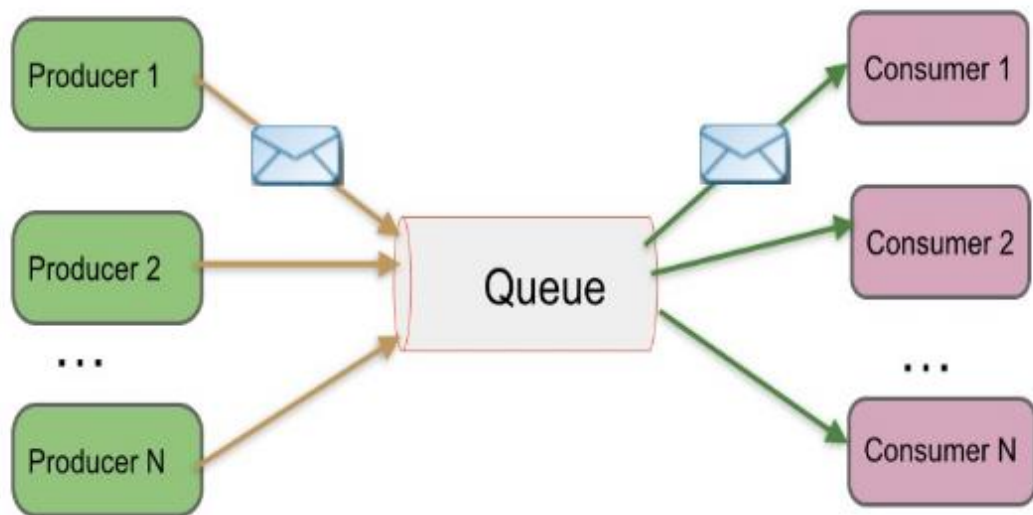
JMS (Java Messaging Service) 是 Java 平台上有关面向消息中间件的技术规范，它便于消息系统中的 Java 应用程序进行消息交换,并且通过提供标准的产生、发送、接收消息的接口，简化企业应用的开发。

## 2 JMS 模型

### 2.1 点对点模型(Point To Point)

生产者发送一条消息到 queue，只有一个消费者能收到。

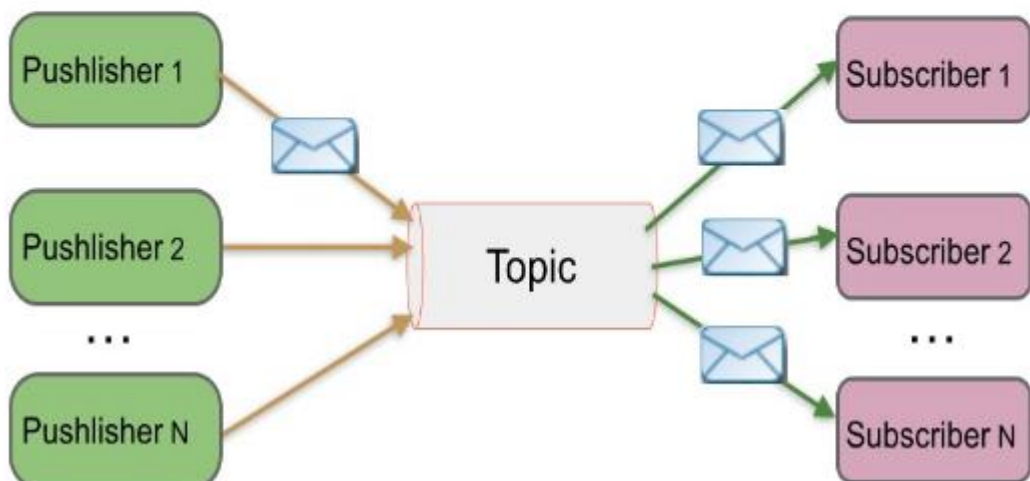
#### 消息队列-点对点



### 2.2 发布订阅模型(Publish/Subscribe)

发布者发送到 topic 的消息，只有订阅了 topic 的订阅者才会收到消息。

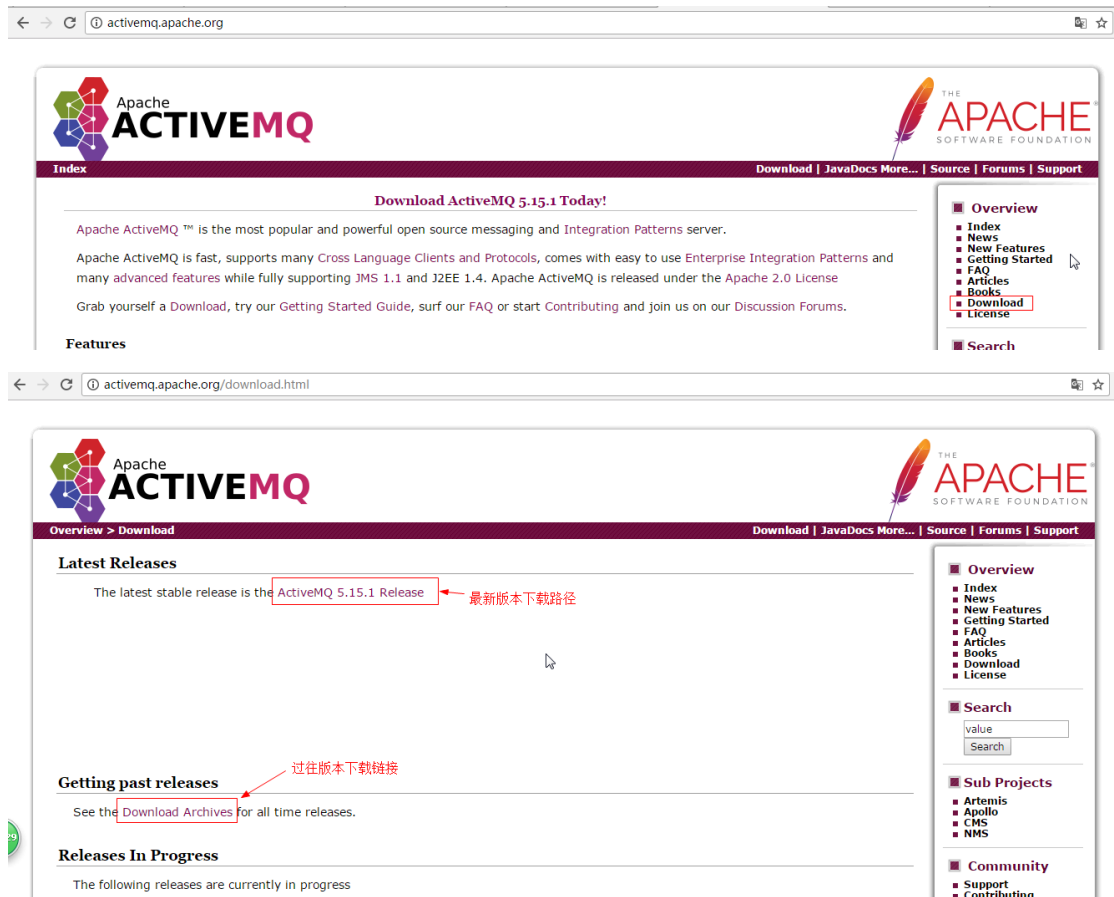
#### 消息队列-发布订阅



## 四、ActiveMQ 安装

### 1 下载资源

ActiveMQ 官网: <http://activemq.apache.org>



#### 1.1 版本说明

ActiveMQ5.10.x 以上版本必须使用 JDK1.8 才能正常使用。

ActiveMQ5.9.x 及以下版本使用 JDK1.7 即可正常使用。

### 2 上传至 Linux 服务器

### 3 解压安装文件

```
tar -zxvf apache-activemq-5.9.0-bin.tar.gz
```

## 4 检查权限

```
ls -al apache-activemq-5.9.0/bin
```

如果权限不足,则无法执行,需要修改文件权限:

```
chmod 755 activemq
```

## 5 复制应用至本地目录

```
cp -r apache-activemq-5.9.0 /usr/local/activemq
```

## 6 启动 ActiveMQ

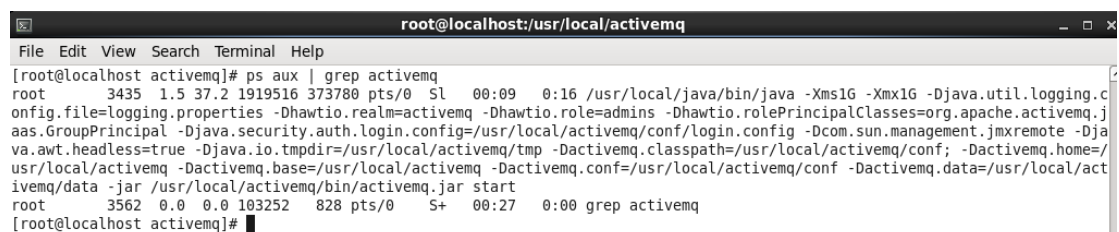
```
/usr/local/activemq/bin/activemq start
```

## 7 测试 ActiveMQ

### 7.1 检查进程

```
ps aux | grep activemq
```

见到下述内容即代表启动成功



```
root@localhost:usr/local/activemq
File Edit View Search Terminal Help
[root@localhost activemq]# ps aux | grep activemq
root      3435  1.5 37.2 1919516 373780 pts/0    Sl   00:09   0:16 /usr/local/java/bin/java -Xms1G -Xmx1G -Djava.util.logging.c
onfig.file=logging.properties -Dhawtio.realm=activemq -Dhawtio.role=admin -Dhawtio.rolePrincipalClasses=org.apache.activemq.j
aas.GroupPrincipal -Djava.security.auth.login.config=/usr/local/activemq/conf/login.config -Dcom.sun.management.jmxremote -Dja
va.awt.headless=true -Djava.io.tmpdir=/usr/local/activemq/tmp -Dactivemq.classpath=/usr/local/activemq/conf; -Dactivemq.home=/
usr/local/activemq -Dactivemq.base=/usr/local/activemq -Dactivemq.conf=/usr/local/activemq/conf -Dactivemq.data=/usr/local/act
ivemq/data -jar /usr/local/activemq/bin/activemq.jar start
root      3562  0.0  0.0 103252   828 pts/0    S+   00:27   0:00 grep activemq
[root@localhost activemq]#
```

### 7.2 管理界面

使用浏览器访问 ActiveMQ 管理应用, 地址如下:

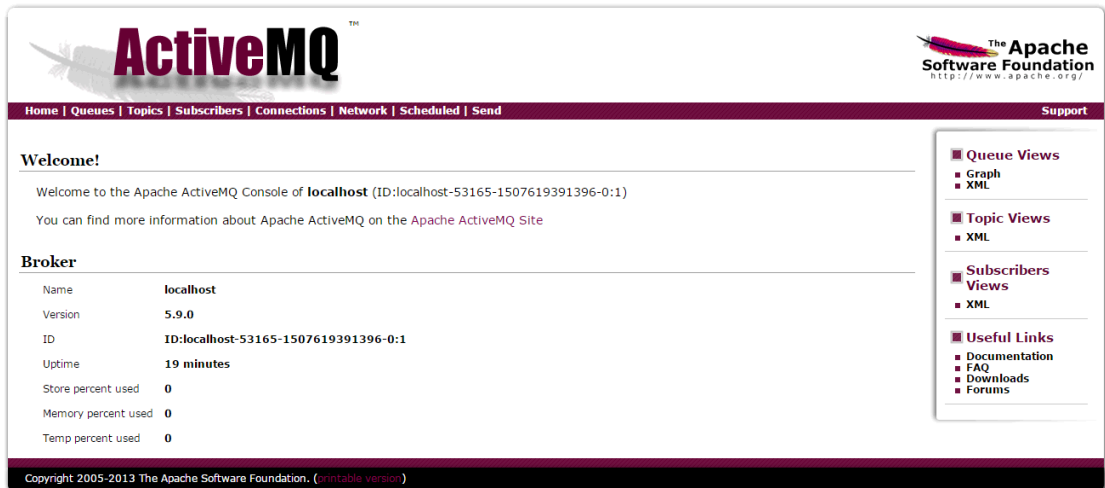
<http://ip:8161/admin/>

用户名: admin

密码: admin

ActiveMQ 使用的是 jetty 提供 HTTP 服务.启动稍慢,建议短暂等待再访问测试.

见到如下界面代表服务启动成功



## 7.3 修改访问端口

修改 ActiveMQ 配置文件: /usr/local/activemq/conf/jetty.xml

```
115 <!-- the default port number for the web console -->
116 <!-- the default port number for the web console -->
117 <!-- the default port number for the web console -->
118 <!-- the default port number for the web console -->
119 <!-- the default port number for the web console -->
120 <!-- the default port number for the web console -->
```

可以修改为其他可用端口

配置文件修改完毕，保存并重新启动 ActiveMQ 服务。

## 7.4 修改用户名和密码

修改 conf/users.properties 配置文件.内容为: 用户名=密码

保存并重启 ActiveMQ 服务即可。

## 8 重启 ActiveMQ

/usr/local/activemq/bin/activemq restart

## 9 关闭 ActiveMQ

/usr/local/activemq/bin/activemq stop

## 10 配置文件 activemq.xml

配置文件中,配置的是 ActiveMQ 的核心配置信息. 是提供服务时使用的配置. 可以修改启动的访问端口. 即 java 编程中访问 ActiveMQ 的访问端口.

默认端口为 61616.

使用协议是: tcp 协议.

修改端口后, 保存并重启 ActiveMQ 服务即可。

---

## 11 ActiveMQ 目录介绍

从它的目录来说,还是很简单的:

- \* bin 存放的是脚本文件
- \* conf 存放的是基本配置文件
- \* data 存放的是日志文件
- \* docs 存放的是说明文档
- \* examples 存放的是简单的实例
- \* lib 存放的是 activemq 所需 jar 包
- \* webapps 用于存放项目的目录

## 五、 ActiveMQ 术语

### 1 Destination

目的地, JMS Provider (消息中间件) 负责维护, 用于对 Message 进行管理的对象。MessageProducer 需要指定 Destination 才能发送消息, MessageReceiver 需要指定 Destination 才能接收消息。

### 2 Producer

消息生成者, 负责发送 Message 到目的地。

### 3 Consumer | Receiver

消息消费者, 负责从目的地中消费【处理|监听|订阅】Message。

### 4 Message

消息, 消息封装一次通信的内容。

## 六、 ActiveMQ 应用

### 1 ActiveMQ 常用 API 简介

下述 API 都是接口类型,由定义在 javax.jms 包中.  
是 JMS 标准接口定义.

#### 1.1 ConnectionFactory

链接工厂, 用于创建链接的工厂类型.



---

## 1.2 Connection

链接. 用于建立访问 ActiveMQ 连接的类型, 由链接工厂创建.

## 1.3 Session

会话, 一次持久有效有状态的访问. 由链接创建.

## 1.4 Destination & Queue

目的地, 用于描述本次访问 ActiveMQ 的消息访问目的地. 即 ActiveMQ 服务中的具体队列. 由会话创建.

interface Queue extends Destination

## 1.5 MessageProducer

消息生成者, 在一次有效会话中, 用于发送消息给 ActiveMQ 服务的工具. 由会话创建.

## 1.6 MessageConsumer

消息消费者【消息订阅者, 消息处理者】, 在一次有效会话中, 用于从 ActiveMQ 服务中获取消息的工具. 由会话创建.

## 1.7 Message

消息, 通过消息生成者向 ActiveMQ 服务发送消息时使用的数据载体对象或消息消费者从 ActiveMQ 服务中获取消息时使用的数据载体对象. 是所有消息【文本消息, 对象消息等】具体类型的顶级接口. 可以通过会话创建或通过会话从 ActiveMQ 服务中获取.

## 2 JMS-HelloWorld

### 2.1 处理文本消息

#### 2.1.1 创建消息生产者

##### 2.1.1.1 创建工程

**New Maven Project**

Configure project

**Artifact**

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

**Parent Project**

Group Id:

Artifact Id:

Version:

► **Advanced**

##### 2.1.1.2 修改 POM 文件添加 ActiveMQ 坐标

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.bjsxt</groupId>
    <artifactId>mq-producer</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <dependencies>
        <!--
https://mvnrepository.com/artifact/org.apache.activemq/activemq-all -->
        <dependency>
            <groupId>org.apache.activemq</groupId>
            <artifactId>activemq-all</artifactId>
            <version>5.9.0</version>
        </dependency>
    </dependencies>

```

#### 2.1.1.3 编写消息的生产者

```
public class HelloWorldProducer {

    /**
     * 生产消息
    */
}
```

```
*/  
  
public void sendHelloWorldActiveMQ(String  
msgTest){  
  
    //定义链接工厂  
    ConnectionFactory connectionFactory = null;  
  
    //定义链接对象  
    Connection connection = null;  
  
    //定义会话  
    Session session = null;  
  
    //目的地  
    Destination destination = null;  
  
    //定义消息的发送者  
    MessageProducer producer = null;  
  
    //定义消息  
    Message message = null;
```

```
try{  
    /**  
        * userName:访问 ActiveMQ 服务的用户名。用户  
        密码。默认的为 admin。用户名可以通过  
jetty-ream.properties 文件进行修改  
        * password:访问 ActiveMQ 服务的用户名。用户  
        密码。默认的为 admin。用户名可以通过  
jetty-ream.properties 文件进行修改  
        * brokerURL:访问 ActiveMQ 服务的路径地址。  
        路径结构为:协议名://主机地址:端口号  
    */  
    connectionFactory = new  
ActiveMQConnectionFactory("admin", "admin",  
"tcp://192.168.70.151:61616");  
  
    //创建连接对象  
    connection =  
connectionFactory.createConnection();  
  
    //启动连接  
    connection.start();  
}
```

```
/**
 * transacted:是否使用事务 可选值为:
true|false
 * true:使用事务 当设置次变量
值。Session.SESSION_TRANSACTED
 * false:不适用事务,设置次变量
则 acknowledgeMode 参数必须设置
 * acknowledgeMode:
 * Session.AUTO_ACKNOWLEDGE:自动消息确认
机制
 * Session.CLIENT_ACKNOWLEDGE:客户端确认
机制
 * Session.DUPS_OK_ACKNOWLEDGE:有副本的客
户端确认消息机制
 */
session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

//创建目的地,目的地名称即队列的名称。消息的
消费者需要通过此名称访问对应的队列
destination =
session.createQueue("helloworld-destination");
```

---

```
        //创建消息的生产者

        producer =
session.createProducer(destination);


        //创建消息对象

        message =
session.createTextMessage(msgTest);


        //发送消息

        producer.send(message);


    }catch(Exception e){
        e.printStackTrace();
    }finally{

        //回收消息发送者资源

        if(producer != null){
            try {
                producer.close();
            } catch (JMSEException e) {

                // TODO Auto-generated catch block
```

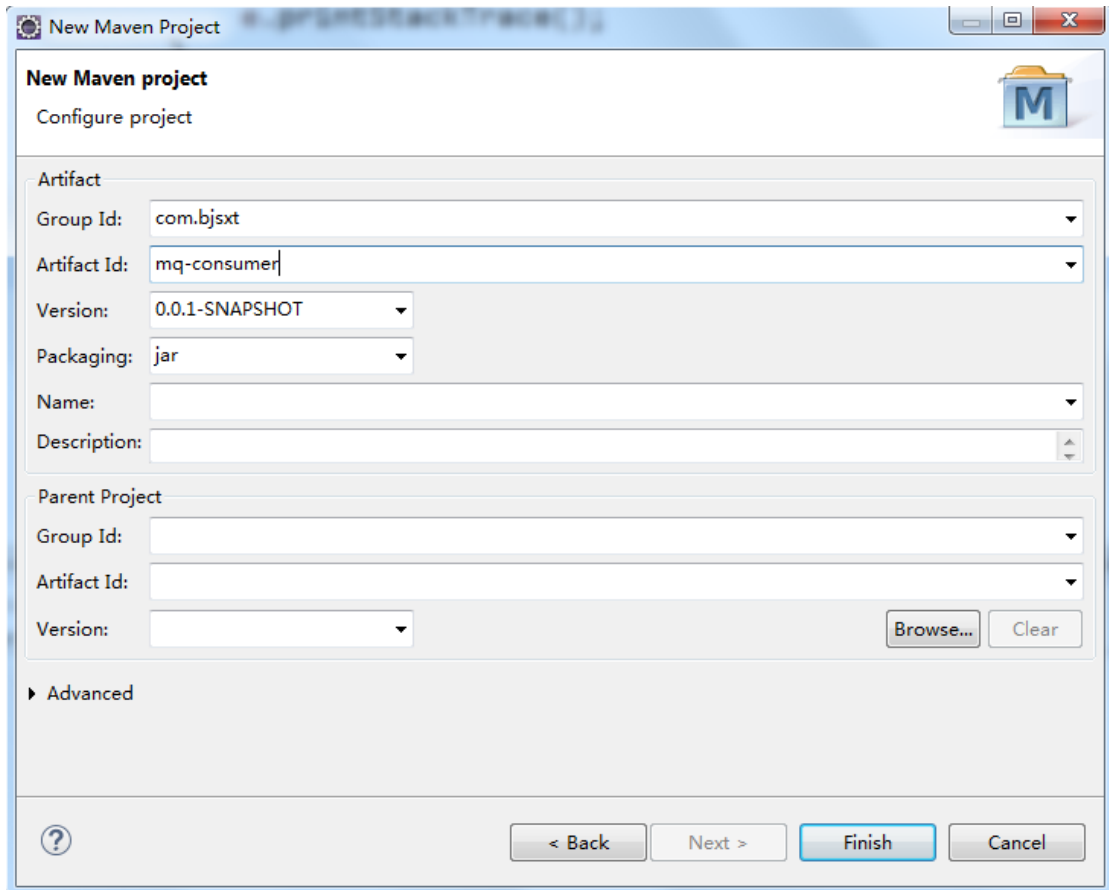
---

```
        e.printStackTrace();
    }
}
if(session != null){
    try {
        session.close();
    } catch (JMSEException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
if(connection != null){
    try {
        connection.close();
    } catch (JMSEException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}
}
```



## 2.1.2 创建消息消费者

### 2.1.2.1 创建工程



New Maven Project

Configure project

Artifact

Group Id: com.bjsxt

Artifact Id: mq-consumer

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Browse... Clear

Advanced

< Back Next > Finish Cancel

### 2.1.2.2 修改 POM 文件添加 ActiveMQ 坐标

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/P
OM/4.0.0
```

```
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.bjsxt</groupId>

    <artifactId>mq-consumer</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <dependencies>

        <!--
https://mvnrepository.com/artifact/org.apache.activemq/activemq-all -->

        <dependency>

            <groupId>org.apache.activemq</groupId>

            <artifactId>activemq-all</artifactId>

            <version>5.9.0</version>

        </dependency>

    </dependencies>

</project>
```

#### 2.1.2.3 编写消息的消费者

```
public class HelloWorldConsumer {

    /**
     * 消费消息
     */
```

```
public void readHelloWorldActiveMQ() {

    // 定义链接工厂
    ConnectionFactory connectionFactory = null;

    // 定义链接对象
    Connection connection = null;

    // 定义会话
    Session session = null;

    // 目的地
    Destination destination = null;

    // 定义消息的发送者
    MessageConsumer consumer = null;

    // 定义消息
    Message message = null;

    try {

        /**
```

---

\* **userName**:访问 **ActiveMQ** 服务的用户名。用户密码。默认的为 admin。用户名可以通过 jetty-ream.

\* **properties** 文件进行修改

\* **password**:访问 **ActiveMQ** 服务的用户名。用户密码。默认的为 admin。用户名可以通过 jetty-ream.

\* **properties** 文件进行修改 **brokerURL**:访问 **ActiveMQ** 服务的路径地址。路径结构为:协议名://主机地址:端口号

\*/

```
connectionFactory = new  
ActiveMQConnectionFactory("admin", "admin",  
"tcp://192.168.70.151:61616");
```

```
// 创建连接对象
```

```
connection =  
connectionFactory.createConnection();
```

```
// 启动连接
```

```
connection.start();
```

```
/**
```

\* **transacted**:是否使用事务 可选值为:

---

`true|false true:使用事务`

`* 当设置次变量值。`

`Session.SESSION_TRANSACTED false:不适用事务,设置次变量`

`* 则 acknowledgeMode 参数必须设置`

`acknowledgeMode:`

`* Session.AUTO_ACKNOWLEDGE:自动消息确认机制`

`* Session.CLIENT_ACKNOWLEDGE:客户端确认机制`

`* Session.DUPS_OK_ACKNOWLEDGE:有副本的客户端确认消息机制`

`*/`

`session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);`

`// 创建目的地，目的地名称即队列的名称。消息的消费者需要通过此名称访问对应的队列`

`destination = session.createQueue("helloworld-destination");`

`// 创建消息的消费者`

```
        consumer =
session.createConsumer(destination);

        // 创建消息对象
        message = consumer.receive();

        //处理消息
        String msg =
((TextMessage)message).getText();

        System.out.println("从 ActiveMQ 服务中获取
的文本信息 "+msg);

    } catch (Exception e) {
        e.printStackTrace();
    } finally {

        // 回收消息发送者资源
        if (consumer != null) {
            try {
                consumer.close();
            } catch (JMSException e) {
                // TODO Auto-generated catch block
            }
        }
    }
}
```

---

```
        e.printStackTrace();
    }
}
if (session != null) {
    try {
        session.close();
    } catch (JMSEException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
if (connection != null) {
    try {
        connection.close();
    } catch (JMSEException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}
}
```

---

### 2.1.3 测试

#### 2.1.3.1 Producer

```
public class Test {  
  
    public static void main(String[] args) {  
        HelloWorldProducer producer = new  
HelloWorldProducer();  
  
        producer.sendHelloWorldActiveMQ("HelloWorld");  
    }  
}
```

#### 2.1.3.2 Consumer

```
public class Test {  
  
    public static void main(String[] args) {  
        HelloWorldConsumer consumer = new  
HelloWorldConsumer();  
  
        consumer.readHelloWorldActiveMQ();  
    }  
}
```



---

```
}
```

## 2.2 处理对象消息

### 2.2.1 定义消息对象

```
public class Users implements Serializable{

    private int userid;
    private String username;
    private int usage;
    public int getUserId() {
        return userid;
    }
    public void setUserId(int userid) {
        this.userid = userid;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public int getUsage() {
```

---

```
        return usage;
    }

    public void setUsage(int usage) {
        this.usage = usage;
    }

    @Override
    public String toString() {
        return "Users [userid=" + userid + ",
username=" + username + ", usage=" + usage + "];"
    }

}
```

### 2.2.2 创建生产者

```
public class HelloWorldProducer2 {

    /**
     * 生产消息
     */

    public void sendHelloWorldActiveMQ(Users
users){
```

---

```
//定义链接工厂
```

```
ConnectionFactory connectionFactory = null;
```

```
//定义链接对象
```

```
Connection connection = null;
```

```
//定义会话
```

```
Session session = null;
```

```
//目的地
```

```
Destination destination = null;
```

```
//定义消息的发送者
```

```
MessageProducer producer = null;
```

```
//定义消息
```

```
Message message = null;
```

```
try{
```

```
    /**
```

```
    * userName:访问 ActiveMQ 服务的用户名。用户
```

---

密码。默认的为 admin。用户名可以通过

jetty-ream.properties 文件进行修改

\* **password**:访问 **ActiveMQ** 服务的用户名。用户  
密码。默认的为 admin。用户名可以通过

jetty-ream.properties 文件进行修改

\* **brokerURL**:访问 **ActiveMQ** 服务的路径地址。  
路径结构为:协议名://主机地址:端口号

```
*/  
  
connectionFactory = new  
ActiveMQConnectionFactory("admin", "admin",  
"tcp://192.168.70.151:61616");
```

```
//创建连接对象  
  
connection =  
connectionFactory.createConnection();
```

```
//启动连接  
  
connection.start();
```

```
/**  
  
* transacted:是否使用事务 可选值为:  
true|false
```

```

        *           true:使用事务 当设置次变量值。
Session.SESSION_TRANSACTED

        *           false:不适用事务,设置次变量
则 acknowledgeMode 参数必须设置

        * acknowledgeMode:

        * Session.AUTO_ACKNOWLEDGE:自动消息确认机
制

        * Session.CLIENT_ACKNOWLEDGE:客户端确认
机制

        * Session.DUPS_OK_ACKNOWLEDGE:有副本的客
户端确认消息机制

    */

    session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

    //创建目的地, 目的地名称即队列的名称。消息的
消费者需要通过此名称访问对应的队列

    destination =
session.createQueue("my-users");

    //创建消息的生产者

    producer =
```

```
session.createProducer(destination);

    //创建消息对象
    message =
session.createObjectMessage(users);

    //发送消息
    producer.send(message);

}catch(Exception e){
    e.printStackTrace();
}finally{

    //回收消息发送者资源
    if(producer != null){
        try {
            producer.close();
        } catch (JMSEException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

```
        if(session != null){
            try {
                session.close();
            } catch (JMSEException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        if(connection != null){
            try {
                connection.close();
            } catch (JMSEException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

### 2.2.3 定义消息消费者

```
public class HelloWorldConsumer2 {
```

---

```
/**
 * 消费消息
 */
public void readHelloWorldActiveMQ() {

    // 定义链接工厂
    ConnectionFactory connectionFactory = null;

    // 定义链接对象
    Connection connection = null;

    // 定义会话
    Session session = null;

    // 目的地
    Destination destination = null;

    // 定义消息的发送者
    MessageConsumer consumer = null;

    // 定义消息
```



```
Message message = null;

try {
    /**
     * userName:访问 ActiveMQ 服务的用户名。用户
    密码。默认的为 admin。用户名可以通过 jetty-ream.
     * properties 文件进行修改
     * password:访问 ActiveMQ 服务的用户名。用户
    密码。默认的为 admin。用户名可以通过 jetty-ream.
     * properties 文件进行修改 brokerURL:访问
    ActiveMQ 服务的路径地址。路径结构为:协议名://主机地址:
    端口号
     */
    connectionFactory = new
    ActiveMQConnectionFactory("admin", "admin",
    "tcp://192.168.70.151:61616");

    // 创建连接对象
    connection =
    connectionFactory.createConnection();

    // 启动连接
```

```
connection.start();

/**
 * transacted:是否使用事务 可选值为:
true|false true:使用事务
 * 当设置次变量值。
Session.SESSION_TRANSACTED false:不适用事务,设置次变
量
 * 则 acknowledgeMode 参数必须设置
acknowledgeMode:
 * Session.AUTO_ACKNOWLEDGE:自动消息确认机
制
 * Session.CLIENT_ACKNOWLEDGE:客户端确认
机制
 * Session.DUPS_OK_ACKNOWLEDGE:有副本的客
户端确认消息机制
 */
session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

// 创建目的地, 目的地名称即队列的名称。消息
的消费者需要通过此名称访问对应的队列
```

```
        destination =  
session.createQueue("my-users");  
  
        // 创建消息的消费者  
        consumer =  
session.createConsumer(destination);  
  
        // 创建消息对象  
        message = consumer.receive();  
  
        //处理消息  
        ObjectMessage objMessage =  
(ObjectMessage)message;  
        Users users =  
(Users)objMessage.getObject();  
        System.out.println(users);  
  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
  
        // 回收消息发送者资源
```

---

```
    if (consumer != null) {
        try {
            consumer.close();
        } catch (JMSEException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    if (session != null) {
        try {
            session.close();
        } catch (JMSEException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    if (connection != null) {
        try {
            connection.close();
        } catch (JMSEException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

```
    }  
    }  
    }  
    }  
}
```

### 3 JMS - 实现队列服务监听

队列服务监听使用的观察者设计模式

#### 3.1 创建消息生产者

```
public class HelloWorldProducer3 {  
  
    /**  
     * 生产消息  
     */  
  
    public void sendHelloWorldActiveMQ(String  
msgTest){  
  
        //定义链接工厂  
        ConnectionFactory connectionFactory = null;  
  
        //定义链接对象
```

---

```
Connection connection = null;
```

```
//定义会话
```

```
Session session = null;
```

```
//目的地
```

```
Destination destination = null;
```

```
//定义消息的发送者
```

```
MessageProducer producer = null;
```

```
//定义消息
```

```
Message message = null;
```

```
try{
```

```
    /**
```

```
        * userName:访问 ActiveMQ 服务的用户名。用户  
        密码。默认的为 admin。用户名可以通过  
jetty-ream.properties 文件进行修改
```

```
        * password:访问 ActiveMQ 服务的用户名。用户  
        密码。默认的为 admin。用户名可以通过  
jetty-ream.properties 文件进行修改
```

```

        * brokerURL:访问 ActiveMQ 服务的路径地址。
路径结构为:协议名://主机地址:端口号

        */

        connectionFactory = new
ActiveMQConnectionFactory("admin", "admin",
"tcp://192.168.70.151:61616");

        //创建连接对象

        connection =
connectionFactory.createConnection();

        //启动连接

        connection.start();

        /**

        * transacted:是否使用事务 可选值为:
true|false

        * true:使用事务 当设置次变量值。
Session.SESSION_TRANSACTED

        * false:不适用事务,设置次变量
则 acknowledgeMode 参数必须设置

        * acknowledgeMode:
```

```
        * Session.AUTO_ACKNOWLEDGE: 自动消息确认机制
        * Session.CLIENT_ACKNOWLEDGE: 客户端确认机制
        * Session.DUPS_OK_ACKNOWLEDGE: 有副本的客户端确认消息机制
    */
    session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

    //创建目的地，目的地名称即队列的名称。消息的消费者需要通过此名称访问对应的队列
    destination =
session.createQueue("my-destination");

    //创建消息的生产者
    producer =
session.createProducer(destination);

    //创建消息对象
    message =
session.createTextMessage(msgTest);
```



```
//发送消息

producer.send(message);

}catch(Exception e){
    e.printStackTrace();
}finally{

    //回收消息发送者资源

    if(producer != null){
        try {
            producer.close();
        } catch (JMSEException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    if(session != null){
        try {
            session.close();
        } catch (JMSEException e) {
            // TODO Auto-generated catch block
```

```
        e.printStackTrace();
    }
}
if(connection != null){
    try {
        connection.close();
    } catch (JMSEException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}
}
}
```

### 3.2消息消费者

```
public class HelloWorldConsumer3 {

    /**
     * 消费消息
     */
}
```

```
public void readHelloWorldActiveMQ() {

    // 定义链接工厂
    ConnectionFactory connectionFactory = null;

    // 定义链接对象
    Connection connection = null;

    // 定义会话
    Session session = null;

    // 目的地
    Destination destination = null;

    // 定义消息的发送者
    MessageConsumer consumer = null;

    // 定义消息
    Message message = null;

    try {
        /**
```

---

\* **userName**:访问 **ActiveMQ** 服务的用户名。用户密码。默认的为 admin。用户名可以通过 jetty-ream.

\* **properties** 文件进行修改

\* **password**:访问 **ActiveMQ** 服务的用户名。用户密码。默认的为 admin。用户名可以通过 jetty-ream.

\* **properties** 文件进行修改 **brokerURL**:访问 **ActiveMQ** 服务的路径地址。路径结构为:协议名://主机地址:端口号

\*/

```
connectionFactory = new  
ActiveMQConnectionFactory("admin", "admin",  
"tcp://192.168.70.151:61616");
```

```
// 创建连接对象
```

```
connection =  
connectionFactory.createConnection();
```

```
// 启动连接
```

```
connection.start();
```

```
/**
```

\* **transacted**:是否使用事务 可选值为:

---

`true|false true:使用事务`

`* 当设置次变量值。`

`Session.SESSION_TRANSACTED false:不适用事务,设置次变量`

`* 则 acknowledgeMode 参数必须设置`

`acknowledgeMode:`

`* Session.AUTO_ACKNOWLEDGE:自动消息确认机制`

`* Session.CLIENT_ACKNOWLEDGE:客户端确认机制`

`* Session.DUPS_OK_ACKNOWLEDGE:有副本的客户端确认消息机制`

`*/`

`session = connection.createSession(false,  
Session.AUTO_ACKNOWLEDGE);`

`// 创建目的地，目的地名称即队列的名称。消息的消费者需要通过此名称访问对应的队列`

`destination =  
session.createQueue("my-destination");`

`// 创建消息的消费者`

```
        consumer =  
session.createConsumer(destination);  
  
        consumer.setMessageListener(new  
MessageListener() {  
  
            //ActiveMQ 回调的方法。通过该方法将消息传  
递到 consumer  
  
            @Override  
            public void onMessage(Message message)  
{  
  
                //处理消息  
  
                String msg=null;  
  
                try {  
  
                    msg =  
((TextMessage)message).getText();  
  
                    } catch (JMSEException e) {  
  
                        // TODO Auto-generated catch block  
                        e.printStackTrace();  
                    }  
  
                    System.out.println("从 ActiveMQ 服务  
中获取的文本信息 "+msg);
```

```
        }  
    });  
  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
}
```

## 4 Topic 模型

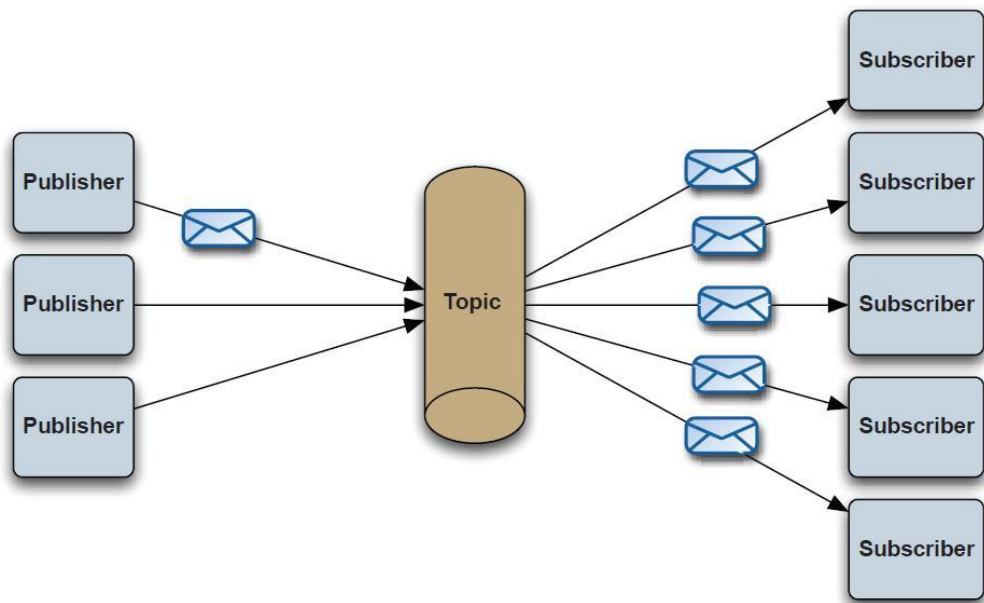
### 4.1 Publish/Subscribe 处理模式（Topic）

消息生产者（发布）将消息发布到 **topic** 中，同时有多个消息消费者（订阅）消费该消息。

和点对点方式不同，发布到 **topic** 的消息会被**所有订阅者消费**。

当生产者发布消息，不管是否有消费者。都不会保存消息

一定要先有消息的消费者，后有消息的生产者。



## 4.2 创建生产者

```
public class HelloWorldProducerTopic {  
  
    /**  
     * 生产消息  
     */  
    public void sendHelloWorldActiveMQ(String  
msgTest){  
  
        //定义链接工厂  
        ConnectionFactory connectionFactory = null;  
  
        //定义链接对象
```



---

```
Connection connection = null;
```

```
//定义会话
```

```
Session session = null;
```

```
//目的地
```

```
Destination destination = null;
```

```
//定义消息的发送者
```

```
MessageProducer producer = null;
```

```
//定义消息
```

```
Message message = null;
```

```
try{
```

```
    /**
```

```
        * userName:访问 ActiveMQ 服务的用户名。用户  
        密码。默认的为 admin。用户名可以通过  
jetty-ream.properties 文件进行修改
```

```
        * password:访问 ActiveMQ 服务的用户名。用户  
        密码。默认的为 admin。用户名可以通过  
jetty-ream.properties 文件进行修改
```

```

        * brokerURL:访问 ActiveMQ 服务的路径地址。
路径结构为:协议名://主机地址:端口号

        */

        connectionFactory = new
ActiveMQConnectionFactory("admin", "admin",
"tcp://192.168.70.151:61616");

        //创建连接对象

        connection =
connectionFactory.createConnection();

        //启动连接

        connection.start();

        /**

        * transacted:是否使用事务 可选值为:
true|false

        *          true:使用事务 当设置次变量值。
Session.SESSION_TRANSACTED

        *          false:不适用事务,设置次变量
则 acknowledgeMode 参数必须设置

        * acknowledgeMode:
```

```
        * Session.AUTO_ACKNOWLEDGE:自动消息确认机制
        * Session.CLIENT_ACKNOWLEDGE:客户端确认机制
        * Session.DUPS_OK_ACKNOWLEDGE:有副本的客户端确认消息机制
    */
    session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

    //创建目的地，目的地名称即队列的名称。消息的消费者需要通过此名称访问对应的队列

    destination =
session.createTopic("test-topic");

    //创建消息的生产者

    producer =
session.createProducer(destination);

    //创建消息对象

    message =
session.createTextMessage(msgTest);
```

---

```
//发送消息

producer.send(message);

}catch(Exception e){
    e.printStackTrace();
}finally{

    //回收消息发送者资源

    if(producer != null){
        try {
            producer.close();
        } catch (JMSEException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    if(session != null){
        try {
            session.close();
        } catch (JMSEException e) {
            // TODO Auto-generated catch block
```

```
        e.printStackTrace();
    }
}
if(connection != null){
    try {
        connection.close();
    } catch (JMSException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}
}
}
```

#### 4.3 创建消费者

```
public class HelloWorldConsumerTopic1 implements
Runnable{

    /**
     * 消费消息
    */
}
```

---

```
*/
```

```
public void readHelloWorldActiveMQ() {
```

```
    // 定义链接工厂
```

```
    ConnectionFactory connectionFactory = null;
```

```
    // 定义链接对象
```

```
    Connection connection = null;
```

```
    // 定义会话
```

```
    Session session = null;
```

```
    // 目的地
```

```
    Destination destination = null;
```

```
    // 定义消息的发送者
```

```
    MessageConsumer consumer = null;
```

```
    // 定义消息
```

```
    Message message = null;
```

```
    try {
```

```
/**
    * userName:访问 ActiveMQ 服务的用户名。用户
    密码。默认的为 admin。用户名可以通过 jetty-ream.
    * properties 文件进行修改
    * password:访问 ActiveMQ 服务的用户名。用户
    密码。默认的为 admin。用户名可以通过 jetty-ream.
    * properties 文件进行修改 brokerURL:访问
    ActiveMQ 服务的路径地址。路径结构为:协议名://主机地址:
    端口号

    */
    connectionFactory = new
    ActiveMQConnectionFactory("admin", "admin",
    "tcp://192.168.70.151:61616");

    // 创建连接对象
    connection =
    connectionFactory.createConnection();

    // 启动连接
    connection.start();

    /**
```

```
        * transacted:是否使用事务 可选值为:  
true|false true:使用事务  
        * 当设置次变量值。  
Session.SESSION_TRANSACTED false:不适用事务,设置次变  
量  
        * 则 acknowledgeMode 参数必须设置  
acknowledgeMode:  
        * Session.AUTO_ACKNOWLEDGE:自动消息确认机  
制  
        * Session.CLIENT_ACKNOWLEDGE:客户端确认  
机制  
        * Session.DUPS_OK_ACKNOWLEDGE:有副本的客  
户端确认消息机制  
        */  
        session = connection.createSession(false,  
Session.AUTO_ACKNOWLEDGE);  
  
        // 创建目的地, 目的地名称即队列的名称。消息  
的消费者需要通过此名称访问对应的队列  
        destination =  
session.createTopic("test-topic");
```



```
// 创建消息的消费者

consumer =
session.createConsumer(destination);

consumer.setMessageListener(new
MessageListener() {

    //ActiveMQ 回调的方法。通过该方法将消息传
    递到 consumer

    @Override
    public void onMessage(Message message)
    {

        //处理消息

        String msg=null;

        try {

            msg =
((TextMessage)message).getText();

        } catch (JMSEException e) {

            // TODO Auto-generated catch block
            e.printStackTrace();

        }

        System.out.println("从 ActiveMQ 服务
```

```
中获取的文本信息 ---topic1 "+msg);

        }

    });

    } catch (Exception e) {

        e.printStackTrace();

    }

}

@Override

public void run() {

    this.readHelloWorldActiveMQ();

}

}
```

## 七、 Spring 整合 ActiveMQ

### 1 创建 spring-activemq-producer

#### 1.1 修改 POM 文件

```
<project

xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://maven.apache.org/P
OM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.bjsxt</groupId>
    <artifactId>parent</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <groupId>com.bjsxt</groupId>
  <artifactId>spring-activemq-producer</artifac
tId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <dependencies>
    <!-- ActiveMQ 客户端完整 jar 包依赖 -->
    <dependency>
      <groupId>org.apache.activemq</groupId>
      <artifactId>activemq-all</artifactId>
    </dependency>
    <!-- ActiveMQ 和 Spring 整合配置文件标签处理 jar
```

---

包依赖 -->

```
<dependency>
    <groupId>org.apache.xbean</groupId>
    <artifactId>xbean-spring</artifactId>
</dependency>

<!-- Spring-JMS 插件相关 jar 包依赖 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
</dependency>

<!-- 单元测试 -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
</dependency>

<!-- 日志处理 -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
</dependency>

<!-- spring -->
```

---

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
</dependency>

<!-- JSP 相关 -->
<dependency>
    <groupId>jstl</groupId>
    <artifactId>jstl</artifactId>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <scope>provided</scope>
```

---

```
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jsp-api</artifactId>
    <scope>provided</scope>
</dependency>
</dependencies>

<build>
    <plugins>
        <!-- 配置 Tomcat 插件 -->
        <plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>
        <configuration>
            <path>/</path>
            <port>8080</port>
        </configuration>
    </plugin>
</plugins>
```

```
</build>

</project>
```

## 1.2 整合 ActiveMQ

```
<?xml version="1.0" encoding="UTF-8"?>

<beans
xmlns="http://www.springframework.org/schema/beans
"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
nstance"
    xmlns:jms="http://www.springframework.org/sch
ema/jms"
    xmlns:context="http://www.springframework.org
/schema/context"
    xmlns:amq="http://activemq.apache.org/schema/
core"
    xsi:schemaLocation="

    http://www.springframework.org/schema/beans

    http://www.springframework.org/schema/beans/spr
ing-beans.xsd
```

---

<http://www.springframework.org/schema/jms>

<http://www.springframework.org/schema/jms/spring-jms.xsd>

<http://activemq.apache.org/schema/core>

<http://activemq.apache.org/schema/core/activemq-core.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context.xsd>">

<!-- 需要创建一个连接工厂,连接 ActiveMQ.  
ActiveMQConnectionFactory. 需要依赖 ActiveMQ 提供的  
amq 标签 -->

<!-- amq:connectionFactory 是 bean 标签的子标签,  
会在 spring 容器中创建一个 bean 对象.

可以为对象命名. 类似: <bean id=""  
class="ActiveMQConnectionFactory"></bean>



```
-->

<amq:connectionFactory
brokerURL="tcp://192.168.70.151:61616"
    userName="admin" password="admin"
id="amqConnectionFactory"/>

<!-- spring 管理 JMS 相关代码的时候,必须依赖 jms 标
签库. spring-jms 提供的标签库. -->

<!-- 定义 Spring-JMS 中的连接工厂对象
    CachingConnectionFactory - spring 框架提供的
连接工厂对象. 不能真正的访问 MOM 容器.

    类似一个工厂的代理对象. 需要提供一个真实工
厂,实现 MOM 容器的连接访问.

-->

<bean id="pooledConnectionFactory"

    class="org.apache.activemq.pool.PooledConnectio
nFactoryBean">

    <property name="connectionFactory"
ref="amqConnectionFactory"></property>

    <property name="maxConnections"
value="10"></property>
```

---

```
</bean>
```

```
<!-- 配置有缓存的 ConnectionFactory，session 的  
缓存大小可定制。 -->
```

```
<bean id="connectionFactory"
```

```
class="org.springframework.jms.connection.CachingConnectionFactory">
```

```
<property name="targetConnectionFactory"  
ref="pooledConnectionFactory"></property>
```

```
<property name="sessionCacheSize"  
value="3"></property>
```

```
</bean>
```

```
<!-- JmsTemplate 配置 -->
```

```
<bean id="template"
```

```
class="org.springframework.jms.core.JmsTemplate">
```

```
<!-- 给定连接工厂，必须是 spring 创建的连接工  
厂。 -->
```

```
<property name="connectionFactory"  
ref="connectionFactory"></property>
```

```
<!-- 可选 - 默认目的地命名 -->
```

```
        <property name="defaultDestinationName"
value="test-spring"></property>

    </bean>

</beans>
```

## 2 创建 spring-activemq-consumer

是一个 jar 工程

### 2.1 修改 POM 文件

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.
0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>com.bjsxt</groupId>
        <artifactId>parent</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <groupId>com.bjsxt</groupId>
```

---

```
<artifactId>spring-activemq-consumer</artifactId>

<version>0.0.1-SNAPSHOT</version>

    <dependencies>

        <!-- activemq 客户端 -->

        <dependency>

            <groupId>org.apache.activemq</groupId>

            <artifactId>activemq-all</artifactId>

        </dependency>

        <!-- spring 框架对 JMS 标准的支持 -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-jms</artifactId>

        </dependency>

        <!-- ActiveMQ 和 spring 整合的插件 -->

        <dependency>

            <groupId>org.apache.xbean</groupId>

            <artifactId>xbean-spring</artifactId>

        </dependency>

    </dependencies>

    <dependency>
```

```
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-beans</artifactId>
</dependency>
</dependencies>
</project>
```

## 2.2 整合 ActiveMQ

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans
"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
nstance"
xmlns:jms="http://www.springframework.org/sch
ema/jms"
xmlns:amq="http://activemq.apache.org/schema/
core"
```

---

```
xsi:schemaLocation="
```

```
http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spr  
ing-beans.xsd
```

```
http://www.springframework.org/schema/jms
```

```
http://www.springframework.org/schema/jms/sprin  
g-jms.xsd
```

```
http://activemq.apache.org/schema/core
```

```
http://activemq.apache.org/schema/core/activemq  
-core.xsd">
```

```
<!-- 需要创建一个连接工厂,连接 ActiveMQ.  
ActiveMQConnectionFactory. 需要依赖 ActiveMQ 提供的  
amq 标签 -->
```

```
<!-- amq:connectionFactory 是 bean 标签的子标签,  
会在 spring 容器中创建一个 bean 对象.
```

```
可以为对象命名. 类似: <bean id=""  
class="ActiveMQConnectionFactory"></bean>
```

```
-->

<amq:connectionFactory
brokerURL="tcp://192.168.70.151:61616"
    userName="admin" password="admin"
id="amqConnectionFactory"/>
```

<!-- spring 管理 JMS 相关代码的时候,必须依赖 [jms](#) 标签库. spring-[jms](#) 提供的标签库. -->

<!-- 定义 Spring-JMS 中的连接工厂对象  
CachingConnectionFactory - spring 框架提供的  
连接工厂对象. 不能真正的访问 MOM 容器.

类似一个工厂的代理对象. 需要提供一个真实工厂,实现 MOM 容器的连接访问.

```
-->

<bean id="connectionFactory"

    class="org.springframework.jms.connection.CachingConnectionFactory">

    <property name="targetConnectionFactory"
ref="amqConnectionFactory"></property>

    <property name="sessionCacheSize"
value="3"></property>
```

---

```
</bean>
```

```
<!-- 注册监听器 -->
```

```
<!-- 开始注册监听.
```

需要的参数有：

`acknowledge` - 消息确认机制

`container-type` - 容器类型 `default|simple`

`simple:SimpleMessageListenerContainer` 最简单的消息监听器容器，只能处理固定数量的 JMS 会话，且不支持事务。

`default:DefaultMessageListenerContainer` 是一个用于异步消息监听器容器，且支持事务

`destination-type` - 目的地类型。使用队列作为目的地。

`connection-factory` - 连接工厂，`spring-jms` 使用的连接工厂，必须是 `spring` 自主创建的

不能使用三方工具创建的工程。如：  
`ActiveMQConnectionFactory`。

```
-->
```

```
<jms:listener-container acknowledge="auto"  
container-type="default"  
destination-type="queue"
```



```
connection-factory="connectionFactory" >

    <!-- 在监听器容器中注册某监听器对象.

         destination - 设置目的地命名

         ref - 指定监听器对象

    -->

    <jms:listener destination="test-spring"
ref="myListener"/>

</jms:listener-container>

</beans>
```

### 3 测试整合

需求:

- 1) 在 producer 中创建 Users 对象
- 2) 将 Users 对象传递到 ActiveMQ 中
- 3) 在 Consumer 中获取 Users 对象并在控制台打印

#### 3.1 Producer 发送消息

##### 3.1.1 如果使用了连接池需要添加两个坐标

###### *PooledConnectionFactoryBean*

```
<dependency>

    <groupId>org.apache.activemq</groupId>

    <artifactId>activemq-pool</artifactId>

    <version>5.9.0</version>

</dependency>
```

```
<dependency>
    <groupId>org.apache.activemq</groupId>

<artifactId>activemq-jms-pool</artifactId>
    <version>5.9.0</version>
</dependency>
```

### 3.1.2 发送消息

```
@Service
public class UserServiceImpl implements
UserService {

    @Autowired
    private JmsTemplate jmsTemplate;

    @Override
    public void addUser(final Users user) {
        //发送消息
        this.jmsTemplate.send(new MessageCreator()
        {

            @Override
```

```
        public Message createMessage(Session
session) throws JMSException {
            Message message =
session.createObjectMessage(user);

            return message;
        }
    });
}

}
```

## 3.2 Consumer 接收消息

### 3.2.1 接收消息

```
@Component(value="myListener")

public class MyMessageListener implements
MessageListener{

    @Autowired

    private UserService userService;

    @Override

    public void onMessage(Message message) {
```

---

```
//处理消息

ObjectMessage objMessage =
(ObjectMessage)message;

Users user=null;

try {
    user = (Users)objMessage.getObject();
} catch (JMSEException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

this.userService.showUser(user);
}
}
```